	Type	L #	Hits	Search Text	DBs	Time Stamp
1	IS&R	L1	871	(712/21,218,241).CCLS.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 15:57
2	BRS	L2	25521437	@ad<"20040412"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 15:57
3	BRS	L3	827	1 and 2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 15:58
4	BRS	L4	97	3 and alias$	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:35
5	BRS	L5	683557	4 and read adj 2 instruction	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 15:58
6	BRS	L6	20	4 and read adj2 instruction	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 15:59

7	BRS	L7	12	6 and write adj2 instruction	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:04

	Type	L #	Hits	Search Text	DBs	Time Stamp
8	BRS	L9	0	7 and dynamic adj identifier and static adj identifier	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:05
9	BRS	L10	0	7 and dynamic adj identifier	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:05
10	BRS	L11	0	7 and static adj identifier	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:05
11	BRS	L8	4	7 and dynamic and identifier	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:06
12	BRS	L12	0	8 and predict*	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:36
13	BRS	L13	6824	alias$ and predict$	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:07

14	BRS	L15	1	14 and displacement	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:36

	Type	L #	Hits	Search Text	DBs	Time Stamp
15	BRS	L14	3	8 and 13	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:09
16	BRS	L16	3	2 and 14	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:09
17	BRS	L17	2	16 and encod$	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:34
18	BRS	L18	0	17 and ("read after write" or RAW)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:34
19	BRS	L19	534502	("read after write" or RAW)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:34
20	BRS	L20	64	3 and 19	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:35

21	BRS	L21	8	20 and alias$	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:35

	Type	L #	Hits	Search Text	DBs	Time Stamp
22	BRS	L22	8	21 not 16	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:35
23	BRS	L23	4	22 and displacement	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:36
24	BRS	L24	0	23 and predict*	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:36
25	BRS	L25	4	23 and predict$	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	2006/05/28 16:36

Terms used
aliasing and **read** and **write** and **displacement** and **rename** and **identifier** and **encodng** and **dynamic** and **predictor** and **repeat**

Sort results by relevance

Display results expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown

Relevance scale

1 The KScalar simulator

J. C. Moure, Dolores I. Rexachs, Emilio Luque
March 2002 **Journal on Educational Resources in Computing (JERIC)**, Volume 2 Issue 1

Publisher: ACM Press

Full text available: pdf(493.35 KB) Additional Information: full citation, abstract, references, index terms

Modern processors increase their performance with complex microarchitectural mechanisms, which makes them more and more difficult to understand and evaluate. KScalar is a graphical simulation tool that facilitates the stuc of such processors. It allows students to analyze the performance behavior of a wide range of processor microarchitectures: from a very simple in-order, scalar pipeline, to a detailed out-of-order, superscalar pipeline with non-blocking caches, speculative execution, and comp ...

Keywords: Education, pipelined processor simulator

2 Euclid and PASCAL

Ted Venema, Jim des Rivieres
March 1978 **ACM SIGPLAN Notices**, Volume 13 Issue 3

Publisher: ACM Press

Full text available: pdf(1.04 MB) Additional Information: full citation, abstract, references, citings

The programming language Euclid was intended for writing system programs that could be verifiable by state-of-the-art verification methods. Since verification was not an explicit goal in the design of Pascal, it is not surprising that this gave rise to differences between the two languages. The Euclid designers intended to change Pascal onl where it fell short of this goal. This paper examines differences in the two languages in the light of this objective. These differences are roughly grouped ...

3 The elements of nature: interactive and realistic techniques

Oliver Deusen, David S. Ebert, Ron Fedkiw, F. Kenton Musgrave, Przemyslaw Prusinkiewicz, Doug Roble, Jos Stam, Jerry Tessendorf
August 2004 **Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04**

Publisher: ACM Press

Full text available: pdf(17.65 MB) Additional Information: full citation, abstract

This updated course on simulating natural phenomena will cover the latest research and production techniques fc simulating most of the elements of nature. The presenters will provide movie production, interactive simulation, and research perspectives on the difficult task of photorealistic modeling, rendering, and animation of natural

phenomena. The course offers a nice balance of the latest interactive graphics hardware-based simulation techniques and the latest physics-based simulation techni ...

4 Draft Proposed: American National Standard—Graphical Kernel System

Technical Committee X3H3 - Computer Graphics
February 1984 **ACM SIGGRAPH Computer Graphics**, Volume 18 Issue SI
Publisher: ACM Press
Full text available: pdf(16.07 MB) Additional Information: full citation

5 Dynamic vectorization: a mechanism for exploiting far-flung ILP in ordinary programs

Sriram Vajapeyam, P. J. Joseph, Tulika Mitra
May 1999 **ACM SIGARCH Computer Architecture News , Proceedings of the 26th annual international symposium on Computer architecture ISCA '99**, Volume 27 Issue 2
Publisher: IEEE Computer Society, ACM Press
Full text available: pdf(103.41 KB) Publisher Site Additional Information: full citation, abstract, references, citings, index terms

Several ILP limit studies indicate the presence of considerable ILP across dynamically far-apart instructions in program execution. This paper proposes a hardware mechanism, *dynamic vectorization (DV),* as a tool for quickly building up a large logical instruction window. Dynamic vectorization converts repetitive dynamic instruction sequences into vector form, enabling the processing of instructions from beyond the corresponding program loop to be overlapped with the loop. This enables vec ...

6 Dependence based prefetching for linked data structures

Amir Roth, Andreas Moshovos, Gurindar S. Sohi
October 1998 **ACM SIGOPS Operating Systems Review , ACM SIGPLAN Notices , Proceedings of the eighth international conference on Architectural support for programming languages and operating systems ASPLOS-VIII**, Volume 32 , 33 Issue 5 , 11
Publisher: ACM Press
Full text available: pdf(1.81 MB) Additional Information: full citation, abstract, references, citings, index terms

We introduce a dynamic scheme that captures the accesspat-terns of linked data structures and can be used to predict future accesses with high accuracy. Our technique exploits the dependence relationships that exist between loads that produce addresses and loads that consume these addresses. By identzj+ing producer-consumer pairs, we construct a compact internal representation for the associated structure and its traversal. To achieve a prefetching eflect, a small prefetch engine speculatively t ...

7 Introducing Ada 9X

John Barnes
November 1993 **ACM SIGAda Ada Letters**, Volume XIII Issue 6
Publisher: ACM Press
Full text available: pdf(4.39 MB) Additional Information: full citation, citings, index terms

8 Superscalar architectures: Reducing the complexity of the register file in dynamic superscalar processors

Rajeev Balasubramonian, Sandhya Dwarkadas, David H. Albonesi
December 2001 **Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture**
Publisher: IEEE Computer Society
Full text available: pdf(1.34 MB) Publisher Site Additional Information: full citation, abstract, references, citings

Dynamic superscalar processors execute multiple instructions out-of-order by looking for independent operations within a large window. The number of physical registers within the processor has a direct impact on the size of

this window as most in-flight instructions require a new physical register at dispatch. A large multi-ported registe file helps improve the instruction-level parallelism (ILP), but may have a detrimental effect on clock speed, especially in future wire-limited technologies. ...

9 The family of concurrent logic programming languages

Ehud Shapiro
September 1989 **ACM Computing Surveys (CSUR)**, Volume 21 Issue 3
Publisher: ACM Press
Full text available: pdf(9.62 MB) Additional Information: full citation, abstract, references, citings, index terms

Concurrent logic languages are high-level programming languages for parallel and distributed systems that offer wide range of both known and novel concurrent programming techniques. Being logic programming languages, they preserve many advantages of the abstract logic programming model, including the logical reading of programs and computations, the convenience of representing data structures with logical terms and manipulating them using unification, and the amenability to metaprogrammin ...

10 Selective, accurate, and timely self-invalidation using last-touch prediction

An-Chow Lai, Babak Falsafi
May 2000 **ACM SIGARCH Computer Architecture News , Proceedings of the 27th annual international symposium on Computer architecture ISCA '00**, Volume 28 Issue 2
Publisher: ACM Press
Full text available: pdf(147.55 KB) Additional Information: full citation, abstract, references, citings, index terms

Communication in cache-coherent distributed shared memory (DSM) often requires invalidating (or writing back) cached copies of a memory block, incurring high overheads. This paper proposes Last-Touch Predictors (LTPs) th learn and predict the "last touch" to a memory block by one processor before the block is accessed and subsequently invalidated by another. By predicting a last-touch and (self-)invalidating the block in advance, an LTP hides the inval ...

11 Real-time shading

Marc Olano, Kurt Akeley, John C. Hart, Wolfgang Heidrich, Michael McCool, Jason L. Mitchell, Randi Rost
August 2004 **Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04**
Publisher: ACM Press
Full text available: pdf(7.39 MB) Additional Information: full citation, abstract

Real-time procedural shading was once seen as a distant dream. When the first version of this course was offere four years ago, real-time shading was possible, but only with one-of-a-kind hardware or by combining the effects of tens to hundreds of rendering passes. Today, almost every new computer comes with graphics hardware capable of interactively executing shaders of thousands to tens of thousands of instructions. This course has beer redesigned to address today's real-time shading capabili ...

12 Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history

Renju Thomas, Manoj Franklin, Chris Wilkerson, Jared Stark
May 2003 **ACM SIGARCH Computer Architecture News , Proceedings of the 30th annual international symposium on Computer architecture ISCA '03**, Volume 31 Issue 2
Publisher: ACM Press
Full text available: pdf(169.91 KB) Additional Information: full citation, abstract, references, citings

Deep pipelines and fast clock rates are necessitating the development of high accuracy, multi-stage branch predictors for future processors. Such a predictor uses a collection of predictors, each of which provides its predictions at a different stage of the pipeline front-end. A simple 1-cycle latency line predictor provides predictions in the first stage, followed in a couple of stages later by predictions from a more accurate global predictor. Finally, one or two stages later, a highly accurat ...

13 The block-based trace cache

Bryan Black, Bohuslav Rychlik, John Paul Shen
May 1999 **ACM SIGARCH Computer Architecture News , Proceedings of the 26th annual international symposium on Computer architecture ISCA '99**, Volume 27 Issue 2
Publisher: IEEE Computer Society, ACM Press
Full text available: pdf(181.08 KB) Publisher Site Additional Information: full citation, abstract, references, citings, index terms

The trace cache is a recently proposed solution to achieving high instruction fetch bandwidth by buffering and reusing dynamic instruction traces. This work presents a new block-based trace cache implementation that can achieve higher IPC performance with more efficient storage of traces. Instead of explicitly storing instructions of a trace, pointers to blocks constituting a trace are stored in a much smaller trace table. The block-based trace cach renames fetch addresses at the basic block le ...

14 Type-Safe linking with recursive DLLs and shared libraries
Dominic Duggan
November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6
Publisher: ACM Press
Full text available: pdf(658.62 KB) Additional Information: full citation, abstract, references, citings, index terms

Component-based programming is an increasingly prevalent theme in software development, motivating the nee for expressive and safe module interconnection languages. Dynamic linking is an important requirement for module interconnection languages, as exemplified by dynamic link libraries (DLLs) and class loaders in operating systems and Java, respectively. A semantics is given for a type-safe module interconnection language that supports shared libraries and dynamic linking, as well as circular ...

Keywords: Dynamic Linking, Module Interconnection Languages, Recursive Modules, Shared Libraries

15 Predictive techniques for aggressive load speculation
Glenn Reinman, Brad Calder
November 1998 **Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture**
Publisher: IEEE Computer Society Press
Full text available: pdf(1.94 MB) Additional Information: full citation, references, citings, index terms

16 Abstract state machines capture parallel algorithms
Andreas Blass, Yuri Gurevich
October 2003 **ACM Transactions on Computational Logic (TOCL)**, Volume 4 Issue 4
Publisher: ACM Press
Full text available: pdf(610.28 KB) Additional Information: full citation, abstract, references, index terms

We give an axiomatic description of parallel, synchronous algorithms. Our main result is that every such algorithr can be simulated, step for step, by an abstract state machine with a background that provides for multisets.

Keywords: ASM thesis, Parallel algorithm, abstract state machine, postulates for parallel computation

17 Memory dependence prediction using store sets
George Z. Chrysos, Joel S. Emer
April 1998 **ACM SIGARCH Computer Architecture News , Proceedings of the 25th annual international symposium on Computer architecture ISCA '98**, Volume 26 Issue 3
Publisher: IEEE Computer Society, ACM Press
Full text available: pdf(1.66 MB) Publisher Site Additional Information: full citation, abstract, references, citings, index terms

For maximum performance, an out-of-order processor must issue load instructions as early as possible, while avoiding memory-order violations with prior store instructions that write to the same memory location. One approach is to use memory dependence prediction to identify the stores upon which a load depends, and communicate that information to the instruction scheduler. We designate the set of stores upon which each load has depended as the load's "store set". The processor can discover and u ...

18 "Flea-flicker" Multipass Pipelining: An Alternative to the High-Power Out-of-Order Offense

Ronald D. Barnes, Shane Ryoo, Wen-mei W. Hwu

November 2005 **Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture MICRO 38**

Publisher: IEEE Computer Society

Full text available: pdf(346.82 KB) Publisher Site

Additional Information: full citation, abstract

As microprocessor designs become increasingly powerand complexity-conscious, future microarchitectures must decrease their reliance on expensive dynamic scheduling structures. While compilers have generally proven adep at planning useful static instruction-level parallelism, relying solely on the compiler¿s instruction execution arrangement performs poorly when cache misses occur, because variable latency is not well tolerated. This paper proposes a new microarchitectural model, multipass pipel ...

19 Link-time binary rewriting techniques for program compaction

Bjorn De Sutter, Bruno De Bus, Koen De Bosschere

September 2005 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 27 Issue 5

Publisher: ACM Press

Full text available: pdf(1.37 MB) Additional Information: full citation, abstract, references, index terms

Small program size is an important requirement for embedded systems with limited amounts of memory. We describe how link-time compaction through binary rewriting can achieve code size reductions of up to 62&percent for statically bound languages such as C, C++, and Fortran, without compromising on performance. We demonstrate how the limited amount of information about a program at link time can be exploited to overcome overhead resulting from separate compilation. This is done with sc ...

Keywords: Program representation, binary rewriting, code abstraction, compaction, interprocedural analysis, linker, whole-program optimization

20 Interprocedural pointer alias analysis

Michael Hind, Michael Burke, Paul Carini, Jong-Deok Choi

July 1999 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 21 Issue 4

Publisher: ACM Press

Full text available: pdf(502.42 KB) Additional Information: full citation, abstract, references, citings, index terms, review

We present practical approximation methods for computing and representing interprocedural aliases for a prograi written in a language that includes pointers, reference parameters, and recursion. We present the following contributions: (1) a framework for interprocedural pointer alias analysis that handles function pointers by constructing the program call graph while alias analysis is being performed; (2) a flow-sensitive interprocedural pointer alias analysis algorithm; (3 ...

Keywords: interprocedural analysis, pointer aliasing, program analysis